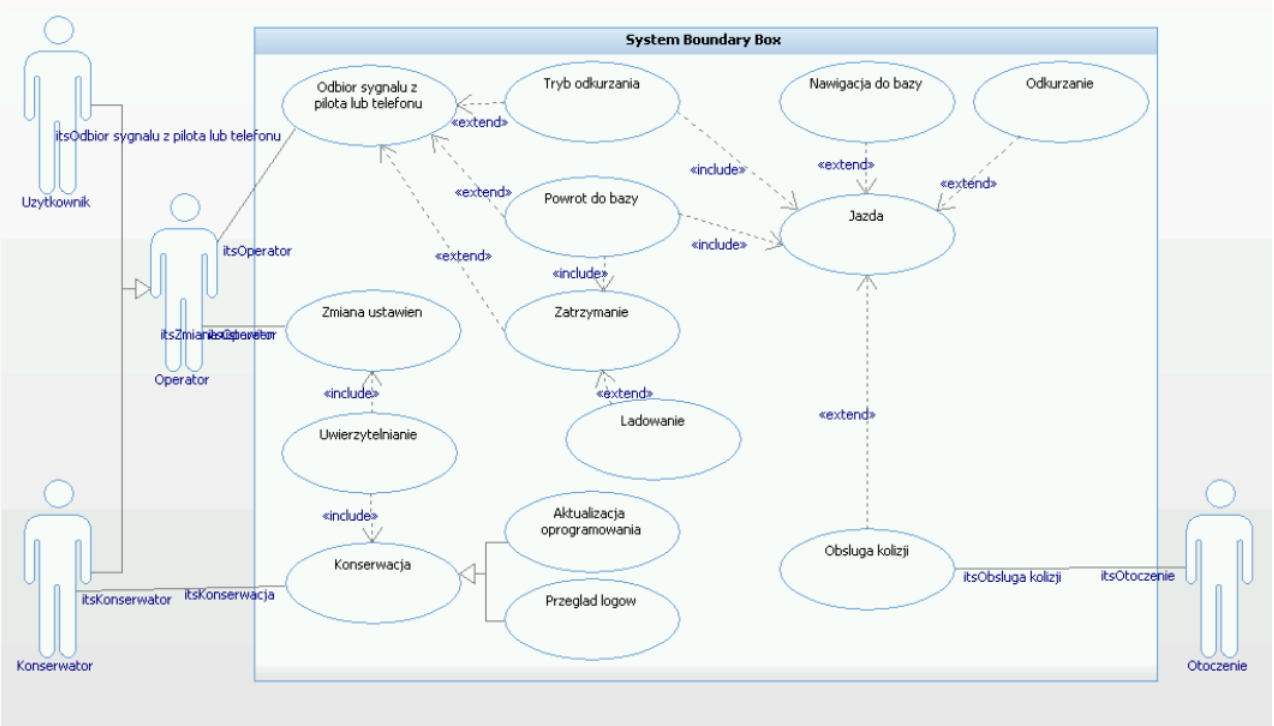
UML, Java

Projekt: odkurzacz typu Roomba

Jan Brzyk

Etap 1

Diagram przypadków użycia funkcji odkurzacza.



Opis:
Zgodnie z instrukcją utworzony został diagram. Po lewej stronie zostali umieszczeni aktorzy- Operator jest uogólnieniem pozostałych aktorów i zostali oni ze sobą połączeni. Po prawej stronie został też dodany aktor “Otoczenie”. W obszarze “System Boundary Box” zostały umieszczone przypadki użycia reprezentujące konkretne funkcje. Zdefiniowano odpowiednie związku pomiędzy przypadkami użycia, dzięki czemu skonstruowano pełny diagram przypadków dla urządzenia.

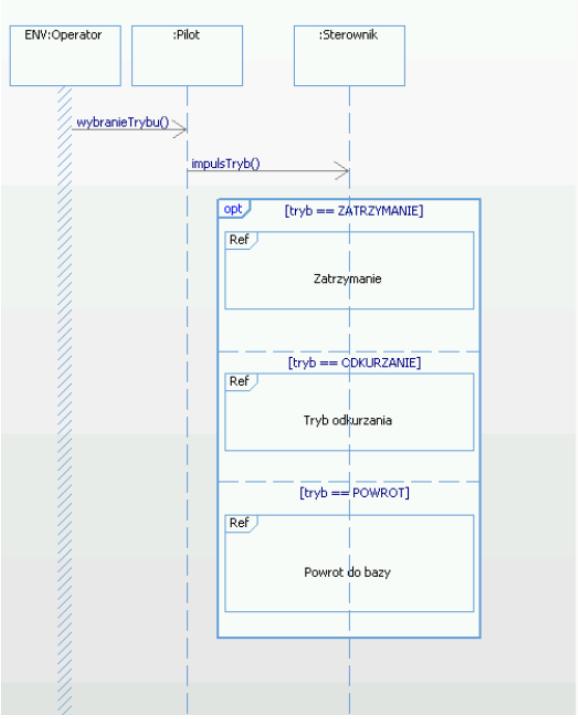
Scenariusz:

Operator, za pomocą aplikacji mobilnej, kontaktuje się z robotem przez sieć wi-fi. W najnowszych rozwiązaniach tego typu operator może wybrać wiele różnych trybów pracy, ustawiając cykliczność pracy odkurzacza, czy polecając robotowi odkurzanie wybranych pokoi. W niniejszym opracowaniu działanie zostało uproszczone, aby pokazać bazową ideę pracy tego urządzenia.
Robot na początku znajduje się w stacji ładowania. Operator może rozpocząć, przerwać oraz zakończyć odkurzanie za pomocą telefonu lub pilota.
Po rozpoczęciu odkurzania robot od razu zaczyna mapować otoczenie za pomocą wbudowanych czujników optycznych. Pozwala to robotowi na unikanie kolizji z meblami i innymi elementami otoczenia, ale przede wszystkim mapowanie (jak sama nazwa wskazuje) tworzy wirtualną mapę odkurzanego pomieszczenia. Pomaga to robotowi zoptymalizować proces i odkurzyć całe pomieszczenie. Możliwe jest dodatkowo wyświetlanie mapy pomieszczenia na urządzeniu mobilnym lub nawet monitorowanie procesu odkurzania w czasie rzeczywistym. Warto również wspomnieć o czujnikach wysokości, dzięki którym robot nie spadnie ze schodów lub większych uskoków, co również jest wpisane w unikanie kolizji.
Po ukończonym odkurzaniu robot wraca do stacji ładowania, aby uzupełnić akumulatory.
Po zakończeniu odkurzania przez operatora robot wraca do stacji ładowania.
Podczas procesu odkurzania mogą się wydarzyć niezaplanowane wypadki, nazwane tutaj **otoczeniem**. Większe przedmioty (kable, małe zabawki) mogą zablokować dalsze działanie elementów czyszczących odkurzacza. Ciekawostką jest fakt, że najnowsze odkurzacze typu Roomba mają wgrane tryby uwalniania z wkręconego w napęd kabla.
Operator jest informowany o wszystkich zdarzeniach zaplanowanych (zakończenie odkurzania, stan baterii itp.) oraz zdarzeniach niezaplanowanych (wszelkiego rodzaju kolizje z otoczeniem i usterki).
Konserwacja to przypadek użycia odkurzacza tylko przez konserwatora (może być to na przykład serwisant w sklepie). Natomiast zwykły użytkownik z założenia nie ma do niego uprawnień. Konserwacja zawiera przegląd logów urządzenia oraz aktualizację oprogramowania.

Etap 2

Diagramy sekwencji

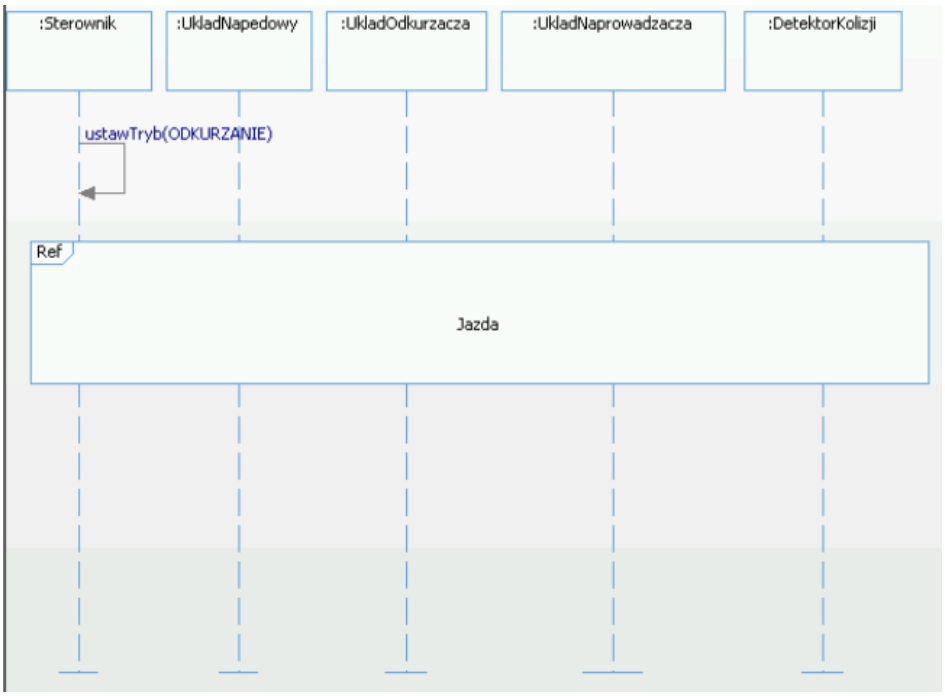
Diagram sekwencji reprezentujący przypadek użycia: Odbiór Impulsu z pilota lub telefonu



W celu uproszczenia sterowania urządzeniem pilotem/telefonem, zastosowano prosty schemat działania. W zależności od obecnego trybu, wysłanie impulsu z pilota/telefonu przez operatora spowoduje przełączenie aktualnej misji odkurzacza w sterowniku.

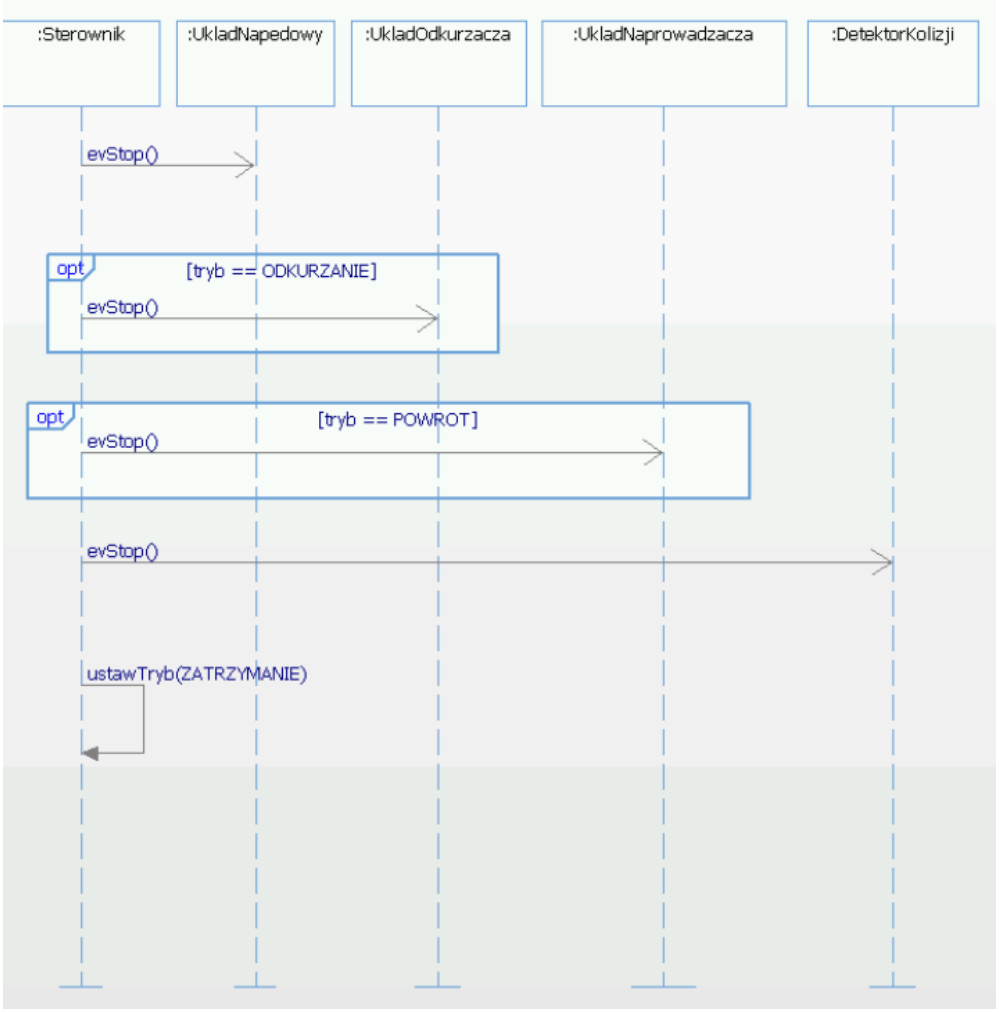
Do diagramu został dodany uogólniony aktor “Operator” oraz utworzone zostały obiekty klas potrzebnych do konkretnej interakcji - “Pilot” i “Sterownik”.
Zamodelowana została współpraca tych dwóch obiektów - komunikat “impulsTryb()” odpowiedzialny za wysłanie impulsu z pilota do sterownika.
Zastosowany został symbol fragmentu wyodrębnionego typu opt, a następnie fragment wyodrębniony został podzielony sekcje symbolem Operand Separator. Każdy, w ten sposób tworzony operand, został opisany odpowiednim warunkiem w kwadratowych nawiasach (np. “[tryb == ODKURZANIE]”.

Diagram sekwencji reprezentujący przypadek użycia: Tryb odkurzania



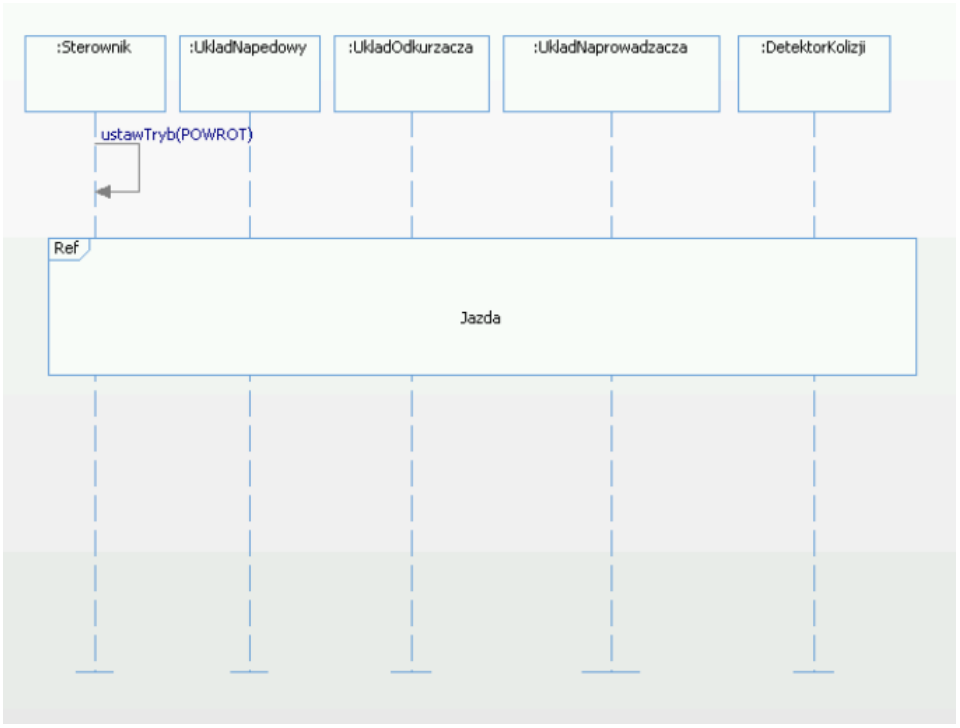
Powodowane jest uruchomienie odkurzania. Stworzenie tego pośredniego przypadku pozwala na dodanie np. instrukcji zjechania z bazy ładowania. Wywoływana zostaje metoda “ustawTryb()” z argumentem “ODKURZANIE”. Jest to realizowane poprzez komunikat samo wywołania. Obecny tu symbol Ref wywołuję “Jazda”.

Diagram sekwencji reprezentujący przypadek użycia: Zatrzymanie



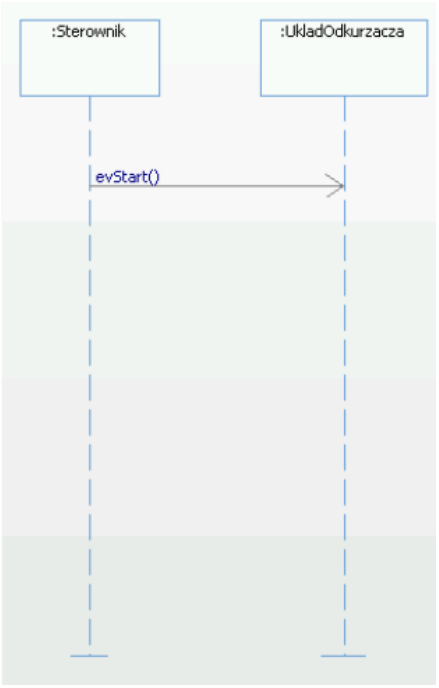
Powodowane jest zatrzymanie elementów urządzenia lub jego powrót do bazy. Dzieje się to poprzez zastosowanie komunikatu “evStop()”, reprezentującego zdarzenie oddziałujące na maszyny stanowe klas “UkladNapędowy”, “UkladOdkurzacza”, “DetektorKolizji” oraz “UkladNaprowadzacza”, powodując ich wyłączenie. Następuje również samo wywołanie “ustawTryb(ZATRZYMANIE)”, zmieniające aktualny tryb działania odkurzacza.

Diagram sekwencji reprezentujący przypadek użycia: Powrót do bazy



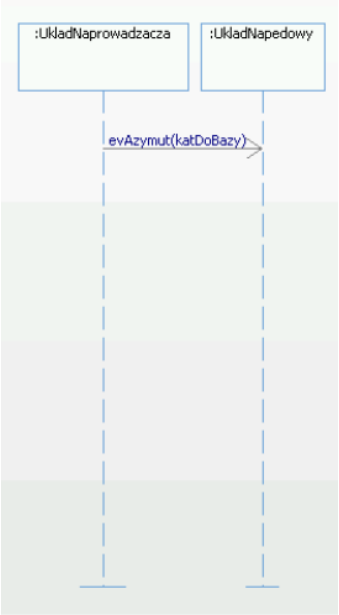
Powodowany jest zjazd do bazy: jazdę na wprost do bazy wg. czujnika.

Diagram sekwencji reprezentujący przypadek użycia: Odkurzanie



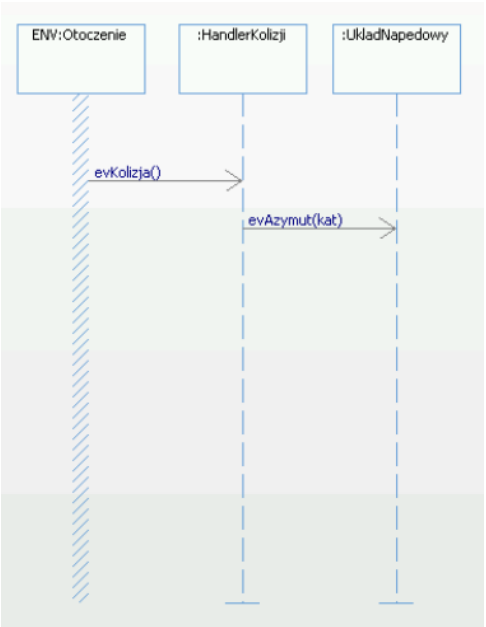
Podczas odkurzania odkurzacz jedzie na wprost, przy kolizji uruchamiana jest sekwencja “UnikanieKolizji”.

Diagram sekwencji reprezentujący przypadek użycia: Nawigacja do bazy



Oddzielna sekwencja dla ładowania pozwala zaimplementować działania specjalne dla momentu ładowania. Po zakończeniu ładowania, zostanie wywołany diagram “WstrzymanieOdkurzania”.

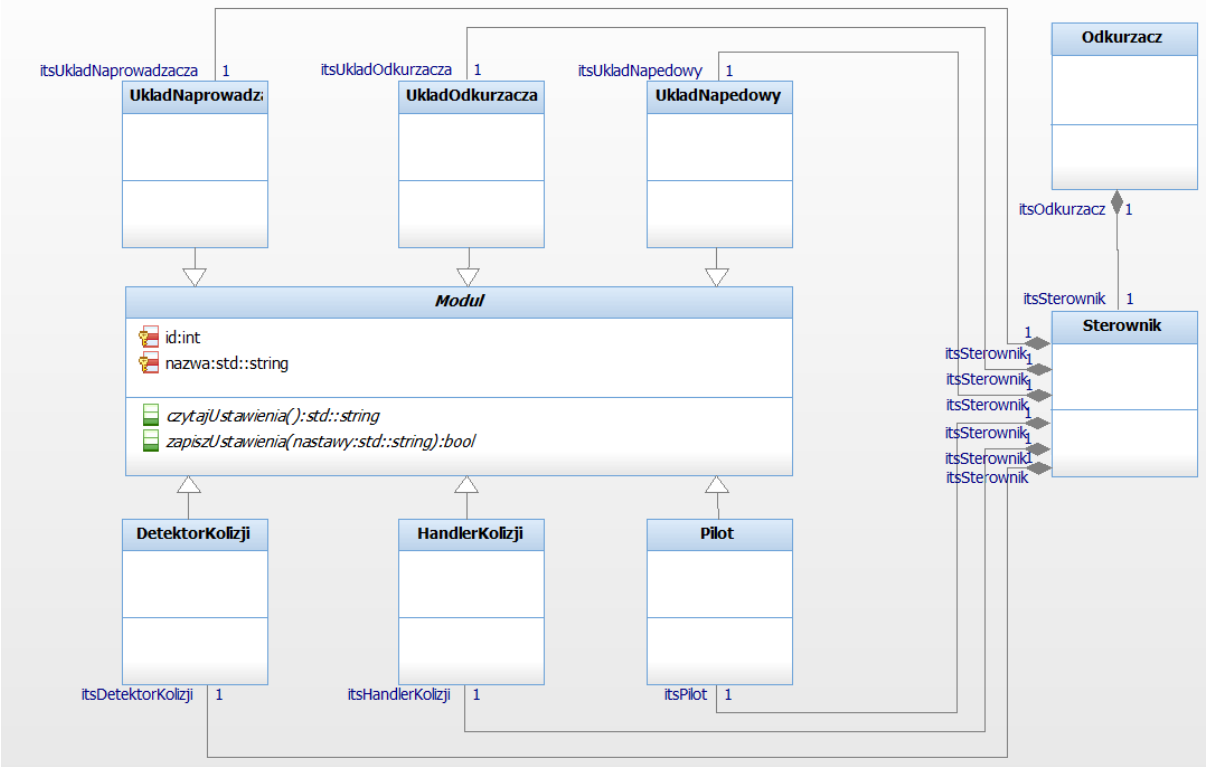
Diagram sekwencji reprezentujący przypadek użycia: Obsługa kolizji



W momencie kolizji urządzenie cofa się o 5 cm, obraca w prawo o 30° i próbuje jechać dalej.

Etap 3

Diagram klas - statyczny diagram utworzony w celu zebrania wszystkich klas i utworzenia związków między nimi.



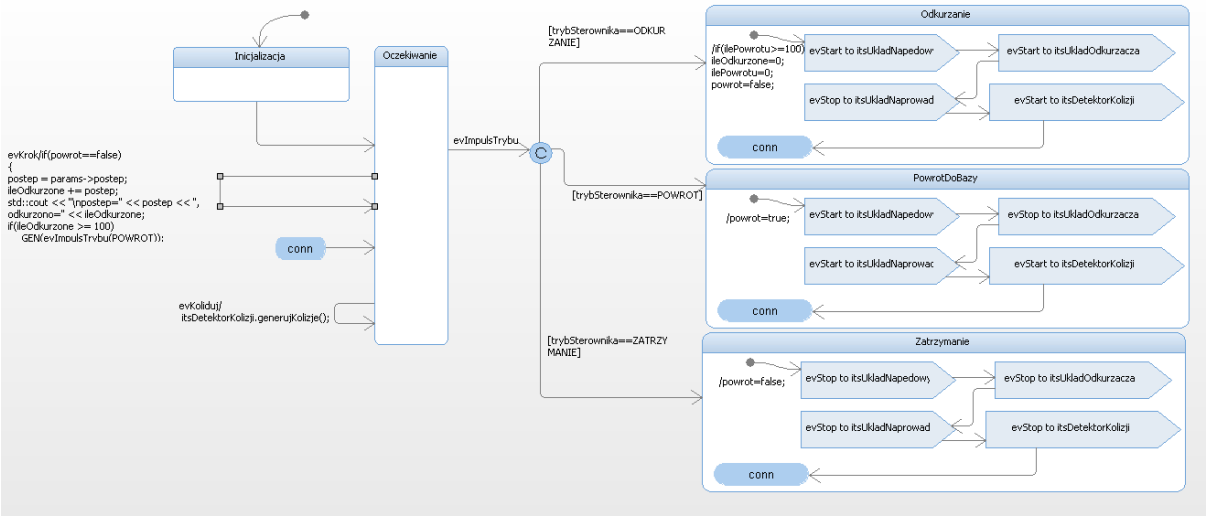
Utworzona została klasa “Odkurzacz” w celu uogólnienia całego modelu. Za pomocą połączenia “Composition” połączono ją z klasą “Sterownik”, a następnie tym samym rodzajem połączenia klasę “Sterownik” ze wszystkimi pozostałymi. Połączenie “Composition” sygnalizuje komponenty/składniki danej klasy (tej, po której stronie znajduje się zamalowany romb).
Kolejna utworzona klasa to “Modul”. Jej istnienie nie wynika z wymagań funkcjonalnych modelu, tylko z idei programowania obiektowego. Dzięki niej klasa “Sterownik” będzie mogła jednolicie posługiwać się wszystkimi modułami. Klasa “Modul” została klasą abstrakcyjną.
Do klasy dodano chronione (bo będą dziedziczone) atrybuty “nazwa” oraz “id” reprezentujące kolejno nazwę modułu i identyfikator. Dodano również operacje “czytajUstawienia” bez argumentów oraz “zapiszUstawienia” z argumentem “nastawy”. Jest to uogólnienie wspólnych parametrów, które będą różne dla każdego modułu. Obie operacje zostały ustawione jako wirtualne i abstrakcyjne.
Za pomocą połączenia “Generalization” wszystkie klasy reprezentujące różne moduły zostały połączone z abstrakcyjną, bazową klasą “Modul”.
Dzięki utworzonym zależnościom, po wywołaniu konstruktora tworzącego klasę “Brama”, zostanie stworzona cała reszta klas i zależności w zbudowanym modelu.

Diagram maszyny stanów (Statechart Diagram)

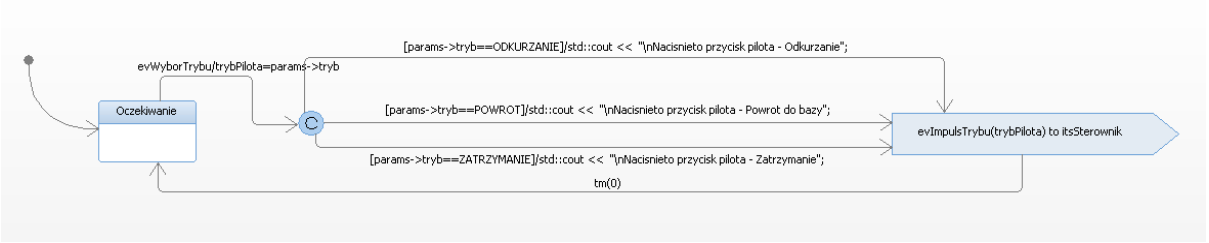
Diagramy maszyny stanowej dotyczą pojedynczych klas i są konstruktywne, czyli, z ich pomocą, Rhapsody potrafi generować kod źródłowy w wybranym języku implementacji (tutaj C++).

Do kolejnych klas dodano Statecharty. Zdefiniowano stany w jakich znajdują się obiekty danej klasy. W odpowiednich miejscach użyto narzędzi do definiowania akcji początkowych (Action on entry), końcowych (Action on exit) lub akcji ciągłych. Akcje mają formę kodu w języku C++. Między stanami muszą się znajdować przejścia (Transitions) ze zdefiniowanymi zdarzeniami wyzwalającymi (Trigger) i opcjonalnie zdefiniowanymi warunkami (Guard) i efektami (Action). To za pomocą tych parametrów operowano eventami w programie. Aby zamodelować oczekiwanie programu przez określony czas użyto w przejściach funkcji timera (np. tm(5)).

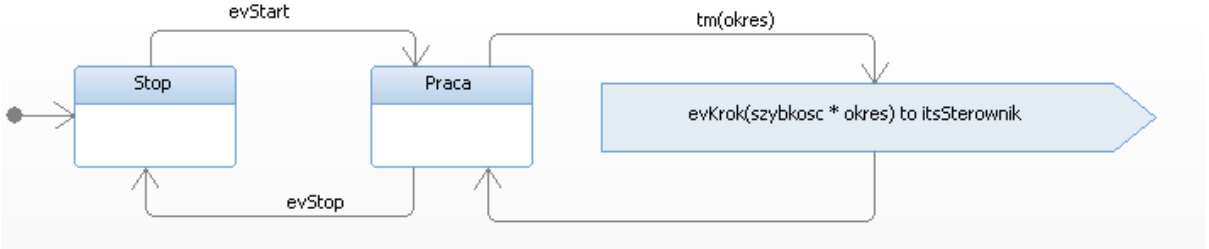
Statechart klasy Sterownik



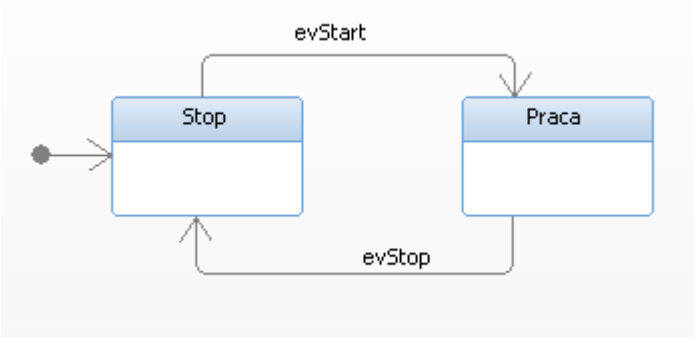
Statechart klasy Pilot



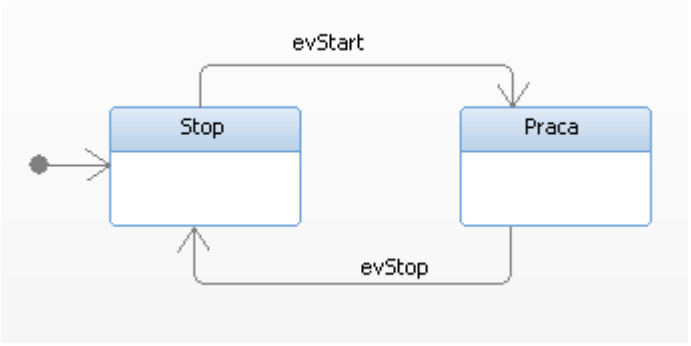
Statechart klasy UkladNapedowy



Statechart klasy UkladOdkurzacza



Statechart klasy UkladNaprowadzacza



Statechart klasy DetektorKolizji

Etap 4

Panel oraz animacja modelu
Animacja jest stosowana do uruchamiania i testowania tworzonego systemu. Pozwala szybciej i łatwiej wychwytywać błędy i prowadzić uogólnione prace uruchomieniowe. Panel (eng. GUI - graphical user interface) pozwala na obserwację działania systemu, niestety nie pozwala na wygodne wywołanie eventów (pushbutton nie pozwala na wyzwolenie eventu z parametrem). Eventy należy wywołać narzędziem Event Generator.

Na panelu miernik wskazówkowy (gauge) pokazuje postęp odkurzania. Poniżej, w polu tekstowym użytkownik/konserwator mogą zobaczyć obecny tryb pracy. Odpowiedni sygnalizatory LED informują o takich wydarzeniach jak: kolizja, cofanie, obracanie się, odkurzanie i włączony napęd. Został również dodany wskaźnik poziomu (level indicator) pokazujący odległość odkurzacza do bazy ładującej. Możliwe jest ręczne generowanie kolizji za pomocą narzędzia Event Generator, w celu zbadania reakcji układu na to wydarzenie.

Podsumowanie

W porównaniu z przykładowym zagadnieniem projektowym omówionym na wykładzie z przedmiotu Inżynieria Oprogramowania - brama garażowa, problem modelowania układu odkurzacza typu Roomba okazał się trudny do rozwiązania. Aby przygotować oprogramowanie w środowisku Rhapsody, które wiernie odzwierciedla wszystkie funkcje, możliwości i parametry tego urządzenia, potrzebne są duże zasoby czasowe i nie podstawowa wiedza. Wspólnie zdecydowano na uproszczenie modelu zachowania odkurzacza, aby efekty końcowy przypominał stopniem skomplikowania projekt bramy z wykładu.
Zdobyte informacje i umiejętności pozwalają na tworzenie wielu innych modelu w środowisku Rhapsody. Wzbogaciła się również nasza wiedza na temat obiektowego podejścia do programowania. Dodatkowo, rozpracowany model odkurzacza może okazać się przydatny przy hobbystycznej budowie własnego.

Źródła:
https://irobot.pl/pl/
https://electronics.howstuffworks.com/gadgets/home/robotic-vacuum2.htm?fbclid=IwAR11ZriHZUsw6RvuWeD593RWYaxcT9ZKD9m2vSgBOSKsNFkS0kl0deouuYY